ECE 364 Prelab 02 Handout

Bash File I/O

August 27, 2018

Name: ____ ID: ee364_____

Passing this lab will satisfy course objective CO1.

Instructions

- Work in your Prelab02 directory
- Copy all files from ∼ee364/labfiles/Prelab02 into your Prelab02 directory. You may use the following command: cp -r ∼ee364/labfiles/Prelab02/* ./
- To submit, commit your files to SVN. We will only grade the version of the file that is in SVN.
- You must meet all *base requirements* in the syllabus to receive any credit.
- To verify your SVN submission, you may run the following sequence of commands:

```
svn update
svn list
```

1 Sorting Processor Performance (50 pts)

Introduction

As part of your prelab, you ran a suite of benchmarks on AMD and Intel processors by changing parameters such as cache sizes and issue widths. You are provided with a file called `simulations.txt` that contains results of processor simulations for the different parameters, in terms of the resulting CPI and execution time. The format of the file is as follows:

```
<Processor name>,<Cache size>,<Issue width>,<CPI>,<Execution time>
```

Implementation Details

You have been assigned the task of sorting results from `simulations.txt` outlined in the five different ways stated below. You decide to write a script called `sort.bash` that meets the given requirements.

1. The script should accept a single argument, a filename.
2. If the correct number of arguments is not provided, print an appropriate message and exit with a return code of 1.
3. If the first argument is a non-existent file, print an error message and exit with a return code of 2.
4. Print the 5 fastest performing CPUs (determined by lowest execution time)
5. Print the 3 most efficient CPUs (determined by lowest CPI)
6. Print all CPUs that have a cache size of 4, in order of increasing execution time.
7. Print the n slowest CPUs (determined by highest execution time), where n should be an integer that you prompt the user. (You may assume that the user will always enter a valid integer.)
8. Print to a file called `sorted_CPI.txt` a list of all AMD CPUs (in order of increasing CPI), followed by all Intel CPUs (in order of increasing CPI) Hint: Don't use numeric sort.

Sample Output

Note: Your output must match the sample output exactly. Your script may be tested with a different data file.

```
$ ./sort.bash
Usage: ./sort.bash <filename>

$ ./sort.bash asdf
Error: asdf does not exist.

$ ./sort.bash simulations.txt
The 5 fastest CPUs:
Intel Core i7,32,16,1.456,4368
Intel Core i7,16,16,1.512,4537
Intel Core i7,8,16,1.625,4875
Intel Core i7,4,16,1.850,5550
Intel Core i7,32,8,1.956,5868

The 3 most efficient CPUs:
Intel Core i7,32,16,1.456,4368
Intel Core i7,16,16,1.512,4537
Intel Core i7,8,16,1.625,4875

The CPUs with cache size 4:
Intel Core i7,4,16,1.850,5550
Intel Core i7,4,8,2.350,7050
AMD Opteron,4,16,2.150,7740
Intel Core i7,4,4,2.600,7800
Intel Core i7,4,2,2.725,8175
Intel Core i7,4,1,2.788,8362
AMD Opteron,4,8,2.650,9540
AMD Opteron,4,4,2.900,10440
AMD Opteron,4,2,3.025,10890
AMD Opteron,4,1,3.088,11115

Enter a value for n: 2
The 2 slowest CPUs:
AMD Opteron,1,1,4.738,17055
AMD Opteron,1,2,4.675,16830

$ cat sorted_CPI.txt
AMD Opteron,32,16,1.669,6007
AMD Opteron,16,16,1.738,6255
AMD Opteron,8,16,1.875,6750
...
...
...
Intel Core i7,1,4,3.950,11850
Intel Core i7,1,2,4.075,12225
Intel Core i7,1,1,4.138,12412
```

2 I/O Redirection (50 pts)

```
SCRIPT NAME:            run.bash
INPUT:                  None
OUTPUT:                 To STDOUT and to file
NUMBER OF ARGUMENTS:    0
ARGUMENTS:              None
RETURN CODE:            0 for success
```

Introduction

You have been provided with a directory called `c-files`, which, as the name states, contains several different .c files. You task is to compile and execute all of these files. You decide to write a short script called run.bash that performs this task.

Implementation Details

Your script must meet the following requirements:

- For each of the files in `c-files`, compile it using the command `gcc -Wall -Werror -std=c99 <filename>` and print the message "`Compiling file <filename>`".
- Direct STDERR to /dev/null. This means that if you get any errors while compiling, suppress them from the terminal and direct them to /dev/null.
- Check the return code of the gcc command to see if compilation succeeded. If it didn't, print the message "`Error: Compilation failed.`"
- However, if gcc was successful, print the message "`Compilation succeeded.`" and execute the compiled code in `a.out`. If execution produces any output, send it to a file called `<filename>.out`, where `<filename>` is the name of the `.c` file you compiled.

Sample Output

The following session gives an example of the output format that is expected.

```
$ ./run.bash
Compiling file err.c... Compilation succeeded.
Compiling file helloWorld.c... Compilation succeeded.
Compiling file sel_sort.c... Compilation succeeded.
Compiling file shift.c... Error: Compilation failed.

$ cat err.out
Running file err.c
$ cat helloWorld.out
Hello World
$ cat sel_sort.out
The array was:
        4       2       32      26      94
        53      3       10      24      17

The sorted array is:
        2       3       4       10      17
        24      26      32      53      94
```